# 第4章 类与对象

类是面向对象程序的基本单位，Java是一种纯面向对象程序设计语言，因此Java程序的所有函数（方法）都是属于某个类的，没有脱离类而单独存在的函数。而C++不是纯面向对象程序设计语言，因此在C++程序中，函数是可以不属于某个类而单独存在的。

封装和数据隐藏、继承、多态是面向对象程序设计的主要特征，类实现了封装与数据隐藏，因此类在Java程序设计中具有重要的作用。

## 4.1 类的定义

### 4.1.1 类与对象的概念

在面向对象程序设计中，如Java语言，构成程序的基本单位是类。将描述一个对象的数据（在面向对象的术语中称为属性），和处理这些数据的函数（在面向对象的术语中称为方法）封装在一起就形成类。类中的大多数数据成员只能用本类中的成员函数进行处理，类通过简单的外部接口与外界联系，这样即使类中的数据结构发生改变，只要类的外部接口不变，使用该类的程序就不需要改变，使得软件开发和维护更加方便。

类相当于一个模板，它描述一类对象的状态和行为。对象是类的一个实例，有特定的状态和行为。

例如汽车是一个类，而具体的某辆汽车就是一个对象。汽车的状态可以用一组属性来描述，比如颜色、车身重量、长、宽、高、排量、最高时速等。汽车的行为是由方法（函数）来描述的，如换挡、加速、刹车、转向等。将汽车的这些属性和和方法封装到一起就形成一个汽车类。

对于某辆具体的汽车，就是一个对象，他具有特定的属性值，比如颜色是黑色、车身重量 1490 kg 、长 4520 mm 、宽 1817 mm、高 1421 mm 、排量2497cc、最高时速 242.0 km/h 。而另外一辆汽车又是另一个汽车对象，有他自己不同的属性值，使得一个对象可以与另一个对象能够区分开来。

### 4.1.2 类的定义

例4.1 定义圆类Circle，有属性半径，另外为了方便计算圆的面积和周长，我们给圆类增加另外一个属性圆周率pi，圆类有计算面积的方法getArea()和计算周长的方法getPerimeter()。

根据题目的要求，类Circle的定义如下：

public class Circle

{

double radius;

double pi=3.142;

double getArea()

{

return pi\*radius\*radius;

}

double getPerimeter()

{

return 2\*pi\*radius;

}

}

Java使用class关键词定义类，类的成员用一对大括号括起来，类中可以包含数据成员，如radius和pi，在面向对象的术语中称其为属性；还可以包含函数成员，如getArea()和getPerimeter()，在面向对象的术语中称其为方法。

有了Circle类，我们就可以定义Circle类的对象了，定义一个带有主方法的测试类Test，在主方法中定义Circle类对象，并为圆的半径赋值，然后再输出圆的面积和周长。

class Test

{

public static void main(String[] args)

{

Circle c1= new Circle();

c1.radius = 10;

System.out.println(c1.getArea());

System.out.println(c1.getPerimeter());

Circle c2= new Circle();

c2.radius = 20;

System.out.println(c2.getArea());

System.out.println(c2.getPerimeter());

c1.radius = 30;

System.out.println(c1.getArea());

System.out.println(c1.getPerimeter());

}

}

在主方法中，首先定义圆类对象c1，然后将其半径赋值为10，再通过调用getArea()方法和getPerimeter()方法分别计算面积和周长并输出。按同样的步骤定义圆类对象c2，并为半径赋值，输出面积和周长。最后三行程序将c1的半径改为30，再输出面积和周长，运行结果如下：

314.2

62.839999999999996

1256.8

125.67999999999999

2827.7999999999997

188.51999999999998

从上面主方法的定义中，我们可以发现定义对象与定义变量类似，因此我们可以将类看成是一种特殊的数据类型，类定义好之后，可以定义任意多个对象，对象的属性值是可以改变的。访问对象的成员通过“.”运算符实现。

练习：设计一个矩形类，有长和宽两个属性，以及计算面积和周长的两个方法，再定义一个Test类，在Test类的主方法中定义矩形类的对象，计算矩形的面积和周长并输出。

上面所定义的圆类，虽然实现了计算面积和周长的功能，但是并没有达到面向对象的要求，面向对象要求实现类的封装，也就是说要尽量隐藏类的内部细节，不允许在类的外部直接访问。例如Circle类中的属性radius和pi就应该被隐藏起来，不允许在类的外面直接饮用。

Circle类定义好之后，在程序的很多地方都可能使用了Circle类，如上面的Test类，后来发现半径这个属性radius需要改名为r，这时就比较麻烦，所有用到radius的地方都需要修改，不利于程序的维护，下面我们通过控制类成员的访问权限来实现数据隐藏。

### 4.1.3 实现数据隐藏

如果将类的成员定义为私有的（private），则在类的外面是不可以访问的。例如将Circle类的属性radius改为私有的，则在Test类的主方法中就不能访问radius属性，语句“c1.radius = 10;”会产生语法错误。

下面修改前面的Circle类，将属性定义为私有的，修改后的Circle类如下：

public class Circle

{

private double radius;

private double pi=3.142;

public double getRadius() {

return radius;

}

public void setRadius(double radius) {

this.radius = radius;

}

public double getArea()

{

return pi\*radius\*radius;

}

public double getPerimeter()

{

return 2\*pi\*radius;

}

}

在类中，定义私有成员的关键字是private，定义公有成员的关键字是public。私有成员只有类本身的方法才可以访问，其他任何类的方法都不可以访问，而公有成员在任何地方都是可以访问的。

将属性radius定义为私有之后，由于在其他类中不能被直接访问，因此我们提供两个公有方法setRadius()和getRadius()，分别为radius赋值和获取radius的值。

下面我们解释一下setRadius()方法中的语句“this.radius = radius;”。

关键词this代表当前对象的引用（后面再详细介绍），也就是说谁调用这个方法，这个this就代表谁，this.radius就是这个对象的半径，这个方法中的语句就是将参数radius的值赋给对象的属性radius。

如果直接写radius = radius，当然是不行的，等号前后两个radius都是参数的radius，不能实现为属性赋值的功能。

当然我们也可以为参数起一个与属性不同的名字，例如改成如下的形式是可以的。

public void setRadius(double r) {

radius = r;

}

由于参数名与属性名不同，这里的radius一定就是属性radius了。当类中有很多属性需要初始化时，使用this的好处就是不用再为每个对应的参数起另外的名字了。

修改后的Test类如下：

class Test

{

public static void main(String[] args)

{

Circle c1= new Circle();

Circle c2= new Circle();

c1.setRadius(10);

c2.setRadius(20);

System.out.println(c1.getArea());

System.out.println(c1.getPerimeter());

System.out.println(c2.getArea());

System.out.println(c2.getPerimeter());

}

}

由于不能直接访问属性radius，在Test类中调用setRadius()方法为对象的属性radius赋值。运行结果如下：

314.2

62.839999999999996

1256.8

125.67999999999999

这时，如果将Circle类的属性radius改为r，则只需要修改Circle类的setRadius()和getRadius()两个方法，而在使用Circle类的任何其他程序都不需做任何要修改，从而提高了程序的可维护性。

在设计类时，通常将类的属性定义为私有的，再为每个私有属性定义一个公有的set方法和get方法，即使以后属性的定义发生变化，但只要set方法和get方法的调用方式不变，使用该类的程序就不需要改动。

练习：将前面练习中的矩形类的属性修改为私有的，再添加公有的get和set方法，实现面积和周长的计算。

## 4.2 构造方法

在前面的Test类中，我们是先定义Circle类的对象，然后再调用setRadius()方法为对象的半径赋值，也就是说在定义圆的对象时，并没有指定圆的半径值（也没有办法指定）。如果希望在定义对象的同时指定其属性值，可以通过为类添加构造方法来实现。

### 4.2.1 构造方法的定义

下面为Circle类添加构造方法，以实现在定义圆的对象时，为该对象指定半径的值。

例4.2 带有加构造方法的Circle类。

下面首先给出给出构造方法和修改后的Test类，然后再详细探讨构造方法的特点。首相在Circle类中添加构造方法如下（其他方法保留不变）：

public class Circle

{

private double radius;

private double pi=3.142;

public Circle(double radius) {

this.radius = radius;

}

…...

}

修改Test类如下：

class Test

{

public static void main(String[] args)

{

Circle c1= new Circle(10);

Circle c2= new Circle(20);

System.out.println(c1.getArea());

System.out.println(c1.getPerimeter());

System.out.println(c2.getArea());

System.out.println(c2.getPerimeter());

}

}

运行结果与前面一样。

有了构造方法之后，就可以在创建对象时，通过构造方法的参数为属性提供初值。构造方法的一般格式为

**访问修饰符 类名（[参数列表]）{**

**语句；**

**}**

与一般方法相比，构造方法有如下几个特点。

（1）构造方法的方法名与类名相同。

（2）不能指定构造方法的返回值类型，注意与没有返回值是不同的。

（3）构造方法是在创建对象时由系统自动调用的，不能通过写程序代码调用。也就是说构造方法只有在创建对象时调用一次，其他时间是不可能再被调用的。

（4）构造方法的作用主要是为属性提供初值，以及对象的初始化处理，因此构造方法通常会有一组参数，用于为属性赋初值。

在用new创建对象时，要为构造方法的参数提供实参。如Circle类的构造方法有一个参数，我们在创建Circle类对象的语句中，就要提供一个实参，具体写法如下：

Circle c1= new Circle(10);

创建对象时提供的实参要与构造方法的参数一一对应，否则会产生错误。

练习：仿照圆类，为矩形类添加构造方法，实现相应的功能。

### 4.2.2 默认的构造方法

只要创建对象，系统就会调用构造方法，但是在例题4.1中并没有为Circle类定义构造方法，为什么在Test类的主方法中就可以创建Circle对象呢？那是因为如果我们没有给类定义构造方法，系统会为类提供一个默认的构造方法，默认构造方法是一个没有参数，什么也不做的构造方法，Circle类的默认构造方法如下。

public Circle() {

}

由于默认的构造方法没有参数，因此在创建对象时也不能提供实参。

一个类没有定义构造方法，则系统自动创建一个默认的构造方法，如果一个类已经定义了构造方法，系统是否还提供一个默认的构造方法呢，通过下面实验，我们可以验证，如果一个类已经定义了构造方法，则系统不再提供默认的构造方法。在例4.2的Test类中加入下面的语句将产生错误。

Circle c1= new Circle(); //错误

因为Circle类定义的构造方法有一个参数，因此创建对象时需要提供一个实参。如果不带实参创建对象，就需要调用没有参数的构造方法，由于Circle类已经没有了默认构造方法，产生了错误。

如果我们既需要指定半径创建圆对象，又需要在不指定半径的情况下创建圆对象，就需要同时提供两个构造方法，修改例4.2的Circle类如下。

public class Circle

{

private double radius;

private double pi=3.142;

public Circle() {

}

public Circle(double radius) {

this.radius = radius;

}

……

}

这个Circle类有两个构造方法，一个构造方法没有参数，一个构造方法有一个double型的参数，显然这两个构造方法是重载关系。这时我们就可以在Test类中用不同的方式创建Circle对象，如下面两行代码。

Circle c0= new Circle();

Circle c1= new Circle(10);

通过方法的重载，我们可以为类定义多个构造方法，从而实现以不同的方式创建对象。

通过调用有参数的构造方法，可以在创建圆的对象时指定半径的值，例如上面将对象c1的半径设置为10，那么调用没有参数的构造方法，半径的值是多少呢？我们可以在创建完对象c0一行代码的后面加入下面一行语句。

System.out.println(c0.getRadius());

程序运行后，该行代码输出结果是0，说明如果构造方法不给属性赋值，则double类型的属性缺省值是0。

下面通过一个例子，测试不同数据类型的属性，如果构造方法不为属性赋值，其缺省值各是什么值。

例4.3 测试不同数据类型属性的缺省值。

设计DefaultValue类，为了方便起见，我们将主方法写在了DefaultValue类中，以便可以直接访问私有属性，代码如下：

public class DefaultValue {

private int i;

private long l;

private short s;

private byte b;

private float f;

private double d;

private boolean bl;

private char c;

public static void main(String[] args){

DefaultValue dv = new DefaultValue();

System.out.println(dv.i);

System.out.println(dv.l);

System.out.println(dv.s);

System.out.println(dv.b);

System.out.println(dv.f);

System.out.println(dv.d);

System.out.println(dv.bl);

System.out.println(dv.c);

}

}

运行结果如下：

0

0

0

0

0.0

0.0

false

\_

在DefaultValue类中，定义了不同数据类型的属性，由于没有定义构造方法，因此使用系统提供的默认构造方法，根据输出结果，数值型属性的缺省值是0，布尔型属性的缺省值是false，字符型属性的缺省值是编码为0的字符。

### 4.2.3 拷贝构造方法

前面所介绍的Circle类构造方法，由一个给属性赋初值的实数参数，通过这个构造方法可以创建一个指定半径的圆。有时我们需要创建一个圆对象，要求这个新创建的圆和另一个已经存在的圆一模一样，使用拷贝构造方法可以实现这一功能。

例4.4 修改Circle类，增加圆心坐标属性，修改构造方法，增加拷贝构造方法，在Test类中使用这些方法。

修改后的Circle类如下：

public class Circle

{

private double radius;

private int x;

private int y;

private double pi=3.142;

public Circle() {

}

public Circle(double radius, int x, int y) {

this.radius = radius;

this.x = x;

this.y = y;

}

public Circle(Circle c) {

this.x = c.x;

this.y = c.y;

this.radius = c.radius;

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

public double getRadius() {

return radius;

}

public void setRadius(double radius) {

this.radius = radius;

}

public double getArea()

{

return pi\*radius\*radius;

}

public double getPerimeter()

{

return 2\*pi\*radius;

}

}

由于类中增加了两个属性，因此在构造方法中相应地增加两个参数，为新增加的属性赋初值，拷贝构造方法的定义如下：

public Circle(Circle c) {

this.x = c.x;

this.y = c.y;

this.radius = c.radius;

}

参数是一个Circle类的引用c（引用的概念稍后再详细讨论），在拷贝构造方法中，将参数c的圆心坐标和半径分别赋给新创建圆的圆心坐标属性和半径属性，使得新创建的圆与参数c一样。

修改后的Test类如下：

class Test

{

public static void main(String[] args)

{

Circle c1= new Circle(10,20,30);

Circle c2= new Circle(20,10,15);

Circle c3= new Circle(c2);

System.out.print(c1.getX() + ", ");

System.out.print(c1.getY() + ", ");

System.out.print(c1.getArea() + ", ");

System.out.println(c1.getPerimeter());

System.out.print(c2.getX() + ", ");

System.out.print(c2.getY() + ", ");

System.out.print(c2.getArea() + ", ");

System.out.println(c2.getPerimeter());

System.out.print(c3.getX() + ", ");

System.out.print(c3.getY() + ", ");

System.out.print(c3.getArea() + ", ");

System.out.println(c3.getPerimeter());

}

}

运行结果如下：

20, 30, 314.2, 62.839999999999996

10, 15, 1256.8, 125.67999999999999

10, 15, 1256.8, 125.67999999999999

在主方法中，首先创建两个对象c1和c2（调用具有三个参数的构造方法），然后再创建c3时，使用c2作为实参，因此会调用拷贝构造方法，使得圆c3与圆c2一模一样。通过后面代码的输出结果，可以验证c3与c2一样。

练习：参照Circle类，为矩形类添加中心坐标，以及拷贝构造方法，在Test类中进行测试。

## 4.3 引用

### 4.3.1 引用的概念

在java程序中，除了基本数据类型，其它类型都是引用数据类型，比如数组、类、以及后面介绍的接口等都是引用类型。基本数据类型与引用类型在内存中的存放是有本质不同的，下面通过具体实例说明引用变量的特点。

例4.5 引用变量与基本数据类型变量的差异。继续使用前面例4.4的Circle类，修改的Test类，修改后的代码如下。

class Test

{

public static void main(String[] args)

{

int i1 = 100;

int i2 = i1;

System.out.println("i1: " + i1);

System.out.println("i2: " + i2);

i2 = 200;

System.out.println("i1: " + i1);

System.out.println("i2: " + i2);

Circle c1= new Circle(10.5,20,30);

Circle c2= c1;

System.out.print("c1: " + c1.getX() + ", ");

System.out.print(c1.getY() + ", ");

System.out.println(c1.getRadius() + ", ");

System.out.print("c2: " + c2.getX() + ", ");

System.out.print(c2.getY() + ", ");

System.out.println(c2.getRadius() + ", ");

c2.setX(100);

c2.setY(200);

c2.setRadius(50);

System.out.print("c1: " + c1.getX() + ", ");

System.out.print(c1.getY() + ", ");

System.out.println(c1.getRadius() + ", ");

System.out.print("c2: " + c2.getX() + ", ");

System.out.print(c2.getY() + ", ");

System.out.println(c2.getRadius() + ", ");

}

}

运行结果如下：

i1: 100

i2: 100

i1: 100

i2: 200

c1: 20, 30, 10.5,

c2: 20, 30, 10.5,

c1: 100, 200, 50.0,

c2: 100, 200, 50.0,

程序中首先定义整型变量i1，并赋初值100，再定义整型变量i2，并将i1的值赋给i2，输出i1和i2的值，显然i1和i2的值都是100。

然后将200赋给变量i2，再输出变量i1和i2的值，这时变量i1仍然是100，变量i2的值是200。

接下来的程序定义Circle类的对象c1，半径初始化为10.5，圆心坐标设置为（20,30），再定义Circle类的对象c2，并将c1赋给c2，输出对象c1和ci2的属性值，这时两个对象的半径和圆心坐标都是一样的。

然后对象c2调用set方法改变其半径和圆心的值，再输出对象c1和ci2的属性值，发现对象c1的属性值也发生了改变，与c2完全一样。

下面我们用图示说明基本数据类型变量与引用变量在内存的存储区别。变量i1和i2的变化情况如图4.1所示，程序中定义的基本数据类型变量，会分配一个内存单元，用于保存变量的值，图左侧显示两个变量原来的值，右侧是将i2的值改变为200后的情况。

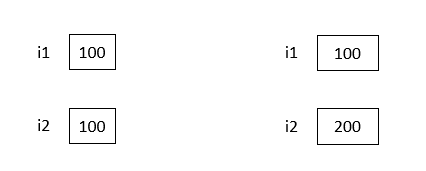


图4.1 基本类型变量在内存的存储

引用变量c1和c2的变化情况如图4.2所示，程序中定义的引用变量，也会分配一个内存单元，但保存的不是对象的具体值，而是用于保存对象的地址。程序中定义引用变量c1，并创建一个Circle对象，将Circle对象的地址保存在引用变量c1中，然后定义引用变量c2，并将c1赋给c2，也就是将c1保存的地址赋给了c2，因此引用变量c2中保存的也是这个Circle对象的地址，也就是说两个引用变量引用的是同一个对象。然后通过c2调用set方法将这个对象的半径、圆心设置为新的值，因此，不论是用c1还是c2访问这个对象，输出结果都是一样的，如图中的右侧。

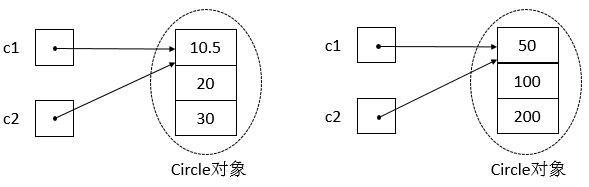


图4.2引用变量在内存的存储

### 4.3.2 this引用

在Java语言中，当创建一个对象后，Java虚拟机就会为其分配一个指向对象本身的指针，这个指针就是this，也称为this引用，注意this引用是与对象相关联的。在程序中恰当使用this，会带来一些方便和好处，下面介绍this的主要用法。

1. 访问对象的属性

当类中的属性与其方法中的参数同名时，或者与方法中的局部变量同名时，为了在该方法中访问类的属性，需要使用this引用。

例如在我们前面定义的Circle类，由于方法setRadius()的参数与属性radius同名，在方法中访问属性radius就要加this指定。由于Java中访问标识符遵循就近原则（指逻辑上的就近，或者说优先访问作用范围最小的标识符），如果不加this指定，该方法访问的一定是参数的radius。而getRadius()方法没有与属性radius同名的参数，因此可以直接访问属性radius。两个函数的代码如下。

public void setRadius(double radius) {

this.radius = radius;

}

public double getRadius() {

return radius;

}

当然在getRadius()方法中，在radius前面使用this限定也是可以的，如下所示。

public double getRadius() {

return this.radius;

}

2. 使用this调用构造方法

在一个类的构造方法内部，也可以使用this关键字调用该类的其它构造方法，这样可以降低代码的重复，也可以使所有的构造方法保持统一，方便以后代码的维护。

下面我们设计一个学生类，体会使用this调用构造方法所带来的好处。假设学生类包括学号、姓名、性别、年龄、身高、体重等属性，这里我们只关心的是构造方法，因此其他方法这里暂不考虑。

例4.6 设计学生类，假设在学生报到时创建学生对象，这时需要为学生分配学号，由于一些特殊原因，某些学生的学号暂时无法确定，只能以后用set方法设置，因此需要设计两个构造方法，一个有学号参数，一个没有学号参数。

为了节省篇幅，下面的代码省去了一组set方法和get方法，只给出一个output()方法，用于输出学生的信息，set方法和get方法请读者自己写出。

public class Student {

private String no;

private String name;

private String gender;

privateint age;

privateint height;

privateint weight;

public Student(String name, String gender, int age, int height, int weight) {

this.name = name;

this.gender = gender;

this.age = age;

this.height = height;

this.weight = weight;

}

public Student(String no, String name, String gender, int age, int height, int weight) {

this(name, gender, age, height,weight);

this.no = no;

}

public void output(){

System.out.print(no + ",");

System.out.print(name + ",");

System.out.print(gender + ",");

System.out.print(age + ",");

System.out.print(height + ",");

System.out.println(weight);

}

}

第二个构造方法使用this调用了第一个构造方法，为除了学号的其他属性初始化，格式如下。

this(name, gender, age, height,weight);

在一个构造方法中调用另一个构造方法的代码必须出现在构造方法的第一行，根据this后面的参数找到匹配的构造方法，如果找不到则产生错误。

编写Test类测试Student类的两个构造方法。

public class Test {

public static void main(String[] args) {

Student s1 = new Student("20170101", "张三", "男", 16, 170,60);

Student s2 = new Student("李四", "女", 17, 165,50);

s1.output();

s2.output();

}

}

运行结构如下：

20170101,张三,男,16,170,60

null,李四,女,17,165,50

创建对象s1时，调用有参数“学号”的构造方法，在该构造方法中，首先通过this关键字调用没有参数“学号”的构造方法，为其他属性初始化，然后再初始化属性“学号”。创建对象s2时，直接调用没有参数“学号”的构造方法，由于没有为属性“学号”赋初值，使用默认值null为其初始化。

3. 在方法中返回对象本身

有时需要方法返回对象本身，可以用“return this”实现，请看下面的例子。

例4.7 设计Person类，包含姓名和年龄两个属性，设计一个增加年龄的方法，编写Test类，测试年龄的增长。

Person类代码如下：

public class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public Person grow(){

age++;

return this;

}

public void output(){

System.out.println(name + ", " + age);

}

}

Test类的代码如下：

public class Test {

public static void main(String[] args) {

Person p = new Person("张三",20);

p.output();

p.grow().grow();

p.output();

}

}

运行结果如下：

张三, 20

张三, 22

Person类的grow()方法将年龄增加1，其返回类型是一个Person对象，因此在方法中使用“return this；”返回对象本身，这样在Test类中，就可以连续多次调用grow()方法。语句“p.grow().grow();”等价于语句“(p.grow()).grow();”，第一次调用grow()方法，返回值是Person对象，因此可以使用这个Person对象继续调用grow()方法。如果将grow()方法定义为：

public void grow(){

age++;

}

则不能用“p.grow().grow();”语句连续调用grow()方法。就只能两次调用grow()方法实现增加2岁的功能，如下两行代码：

p.grow();

p.grow();

## 4.4 类的聚集

聚集是指两个对象之间的归属关系（即has-a关系），也就是一个对象包含另一个对象。例如圆类对象和点类对象之间的关系，点类包含两个属性表示点的坐标，圆类有圆心属性和半径属性，显然圆心可用一个点来表示，这样圆类对象就包含一个点类对象。归属关系中的所有者对象称为聚集对象，而它的类称为聚集类，如刚才所说的圆类。归属关系中的从属对象称为被聚集对象，而它的类被称为被聚集类，如刚才所说的点类。

例4.8 重新设计Circle类，其圆心用一个点类对象表示。

首先设计点类，代码如下：

public class Point {

private int x;

private int y;

public Point() {

}

public Point(int x, int y) {

this.x = x;

this.y = y;

}

public void setX(int x) {

this.x = x;

}

Public int getX() {

return x;

}

public void setY(int y) {

this.y = y;

}

Public int getY() {

return y;

}

}

Point类很简单，包括代表坐标的两个属性，两个构造方法，以及两个属性对应的get、set方法。

修改Circle类，将圆心用Point对象表示，代码如下：

public class Circle {

private Point center;

private double radius;

private double pi = 3.14;

public Circle() {

}

public Circle(int x, int y, double radius) {

this.center = new Point(x,y);

this.radius = radius;

}

public Circle(Point center, double radius) {

this.center = center;

this.radius = radius;

}

public Point getCenter() {

return center;

}

public void setCenter(Point center) {

this.center = center;

}

public double getRadius() {

return radius;

}

public void setRadius(double radius) {

this.radius = radius;

}

public double getArea()

{

return pi\*radius\*radius;

}

public double getPerimeter()

{

return 2\*pi\*radius;

}

}

Circle类的圆心由原来的两个整型属性表示改为由一个Point对象表示，相应的修改了构造方法，除了不带参数的构造方法，还提供了两个构造方法，其中一个构造方法有两个整型参数和一个实型参数，根据两个整型参数创建一个Point对象，并赋给圆心center，将实型参数赋给半径。

另一个构造方法有一个Point类型的引用参数和一个实型参数，Point引用参数赋给圆心，将实型参数赋给半径。

设计Test类，测试Point类和Circle类，代码如下。

public class Test {

public static void main(String[] args) {

Circle c = new Circle(100,200,100.5);

System.out.println(c.getCenter().getX());

System.out.println(c.getCenter().getY());

System.out.println(c.getRadius());

System.out.println(c.getArea());

System.out.println(c.getPerimeter());

}

}

运行结果如下：

100

200

100.5

31714.785

631.14

在创建Circle对象时，为构造方法提供三个实参，因此调用有三个参数的构造方法，在Circle类的构造方法中再创建Point对象，作为圆心。

表达式“c.getCenter().getX()”可以理解为，圆类对象c调用getCenter()方法得到圆心点（Point对象），Point对象再调用getX()方法得到圆心的x坐标。

当然我们也可以先创建Point对象，然后使用俩个参数的构造方法创建Circle对象。我们再修改Test类，代码如下：

public class Test {

public static void main(String[] args) {

Point p = new Point(100,200);

Circle c = new Circle(p,100.5);

System.out.print(c.getCenter().getX() + ",");

System.out.print(c.getCenter().getY() + ",");

System.out.println(c.getRadius());

p.setX(10);

p.setY(20);

System.out.print(c.getCenter().getX() + ",");

System.out.print(c.getCenter().getY() + ",");

System.out.println(c.getRadius());

}

}

运行结果如下：

100,200,100.5

10,20,100.5

需要注意的是，用这种方式创建的Circle对象，由于c的圆心center和Point对象p引用的是同一个对象，当使用p调用set方法改变其坐标时，c的圆心也同时被改变了。例如下面两行代码，就会改变圆c的圆心。

p.setX(10);

p.setY(20);

我们可用图4.3表示这种情况。

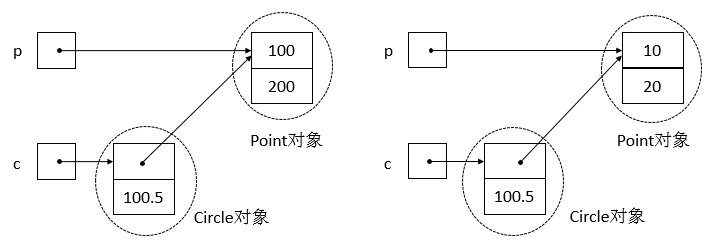


图4.3 圆心坐标被改变

Circle类的对象c在自己不知道的情况下，圆心被修改了，为了避免这种情况的发生，可以用下面的语句创建Circle对象。

Circle c = new Circle(new Point(100,200),100.5);

第一个参数改为“new Point(100,200)”，也就是临时创建一个Point对象作为第一个参数，这个临时对象并没有赋给一个引用变量，我们称之为匿名对象。由于没有定义这个对象的引用变量，除了通过Circle类的引用变量c，就没有其他方式可以改变这个Point对象的值了，避免了在c不知情的情况下改变圆心坐标。可以通过下面的语句改变c的圆心坐标。

c.getCenter().setX(30);

c.getCenter().setY(40);

首先调用Circle类的getCenter()方法获得圆心（Point类的引用），再调用Point类的setX()方法和setY()方法改变圆心的x坐标和y坐标。

## 4.5 静态成员与常量

### 4.5.1 常量

1. 常量属性

在程序运行过程中，值保持不变的量称为常量，Java中用关键字final定义常量。例如前面定义的圆类Circle，其属性pi就是一个常量，定义方法如下所示。

public class Circle

{

private double radius;

private int x;

private int y;

private final double PI=3.14;

……

}

属性前面带有final关键字，表示该属性是常量属性，其值是不可以改变的，因此常量属性必须要初始化。在Java中，常量一般用大写字母。

常量属性的初值可以在定义时给出，如前面定义的PI，也可以在构造方法中为常量属性赋初值，如下面的方式。

public Circle(double radius, int x, int y) {

this.radius = radius;

this.x = x;

this.y = y;

PI = 3.14;

}

注意：常量属性只能初始化一次，如果在定义时已经有了初值，则在构造方法中就不能再初始化，否则产生错误。

2. 方法中定义常量

不仅可以定义常量属性，也可以在方法中使用关键字final定义局部常量，常量可以在定义时初始化，也可以在使用前赋值，但只能赋值一次，试图给常量第二次赋值，则产生语法错误。例如下面的程序，在给常量A和B第二次赋值时产生了错误。

public class LocalFinal {

public static void main(String[] args) {

fina lint A = 10;

final int B;

A=30;//错误，A已经初始化，不能再次赋值

B=40;

B=20;//错误，B已经赋值一次，不能再次赋值

}

}

### 4.5.2 静态成员

1. 静态属性

类中定义的属性，在该类的不同对象中有不同的值。如前面定义的圆类，每个圆都有自己的半径值。但在类中也有一些特殊的属性，该属性的值对于该类的每个对象都是一样的，如圆类中的圆周率，所有的圆都有相同的圆周率。我们可以用关键词static将圆周率这样的属性定义为静态的属性。将属性定义为静态属性，表示该属性值是属于类的，不依赖于任何对象。

如果我们需要记录某各类创建对象的个数，或者实现对象的自动编号，可以使用静态属性。

例4.9 设计学生类，属性有学号、姓名、年龄和一个用于记录对象个数的静态属性count，利用属性count的值实现学号的自动增加。

Student类定义如下：

public class Student {

private static int count=0;

private String number;

private String name;

private int age;

public Student(String name, int age) {

count++;

if(count<10){

this.number = "20170" + count;

}

else{

this.number = "2017" + count;

}

this.name = name;

this.age = age;

}

public String getNumber() {

return number;

}

public void setNumber(String number) {

this.number = number;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

Public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

我们假设学生人数少于100，学号的前四位表示年，后两位表示序号，将静态属性count初始化为0，在构造方法中首先将count值增加1，然后生成6位学号，赋值给number。Student类除了构造方法，只有三个属性的set和get方法。

设计StudentTest类对Student类进行测试，代码如下：

public class StudentTest {

public static void main(String[] args) {

Student s1 = new Student("Zhangsan", 18);

Student s2 = new Student("Lisi", 19);

Student s3 = new Student("Wangwu",17);

System.out.println(s1.getNumber() + "," + s1.getName() + s1.getAge());

System.out.println(s2.getNumber() + "," + s2.getName() + s2.getAge());

System.out.println(s3.getNumber() + "," + s3.getName() + s3.getAge());

}

}

在主方法中定义三个Student对象，然后输出对象的属性值，运行结果如下：

201701,Zhangsan18

201702,Lisi19

201703,Wangwu17

通过运行结果，可以看到学号已经实现了自动增加。

2. 静态方法

前面已经介绍，静态属性是属于类的，不依赖于具体对象，因此静态属性可以通过类名访问，格式是“类名**.**静态属性”，例如“Student.count”。但是由于count是Student类的私有成员，在另一个类中是不能访问的，如果将count改为公有成员，是可以这样访问的。

但是为了类的封装性和数据隐藏，通常将属性声明为私有的，这时为了访问这些私有的静态属性，通常的做法是定义公有的静态方法。可以通过类名调用静态方法实现对私有静态属性的访问。

为Student类添加静态方法getCount()和setCount()，代码如下

public static int getCount() {

return count;

}

public static void setCount(int count) {

Student.count = count;

}

这时我们就可以在StudentTest类的主方法中调用这两个方法实现对静态属性count的访问，修改主方法如下：

public static void main(String[] args) {

System.out.print(Student.getCount()+" ");

Student s1 = new Student("Zhangsan", 18);

System.out.print(Student.getCount()+" ");

Student s2 = new Student("Lisi", 19);

System.out.print(Student.getCount()+" ");

Student s3 = new Student("Wangwu",17);

System.out.println(Student.getCount());

}

运行结果如下：

0 1 2 3

在测试类的主方法中使用类名调用静态方法getCount()获得属性count的值。

注意1：使用类名可以调用静态方法，但不能调用非静态方法。因为类中的非静态成员是属于对象的，只有对象才可以访问非静态成员。

注意2：用静态方法只能访问静态属性或调用静态方法，不能访问非静态属性，也不能调用非静态方法。因为如果可以的话，类就可以间接地访问非静态属性了，显然是不合理的。

也就是说通过类名只能访问到静态属性，不管用什么手段（直接的或间接的）都不能访问到非静态属性。

另外由于静态方法可以被类调用，因此静态方法中没有this引用（this是与对象关联的）。

对象既可以调用非静态方法，也可以调用静态方法。但为了更符合逻辑，通常都使用类名调用静态方法。

3. 静态常量属性

很多时候，类中的静态属性都被定义为常量。例如在圆类中，不仅任何一个圆都具有相同的圆周率，而且圆周率也是一个常量。我们可以将圆类中的圆周率定为静态常量，定义方法如下：

public class Circle

{

private double radius;

public static final double PI=3.14;

……

}

静态常量必须在定义时初始化。

事实上，在Java提供的数学类Math中定义了很多数学常量，其中就有圆周率PI，因此我们不必定义自己的PI，直接使用Math.PI就可以了。

4. 静态代码块

一个类可以使用不包含在任何方法中的静态代码块，静态代码块的格式如下：

static {

……

}

Java的.class字节码文件要想执行，首先要加载到内存，由类加载器把字节码文件的代码加载到内存中，这一过程叫类加载。静态代码块就是在类加载时被执行的，且只被执行一次，静态代码块常用来完成类属性的初始化。

除了静态代码块，java的类中也可以有非静态代码块，格式如下：

{

……

}

非静态代码块是在创建对象时，在执行构造方法之前执行的，每创建一个对象都要执行一次。

下面以一个人出生后要建立档案为例，介绍静态代码块、非静态代码块和构造方法的执行顺序。

例4.10 设计Person类，用于出生建立档案，为了简单起见，只涉及有三个属性，国籍、年龄和姓名。由于在国内出生，国籍都是“中国”，因此可以将其定义为静态常量；出生时年龄都是0岁（周岁），每个对象都一样，但将来每个人的年龄都是可以单独变化的；姓名在创建对象时每个人都是不一样的。

Person类代码如下：

public class Person {

public static final String nationality;

private int age;

private String name;

static {

nationality = "中国";

System.out.println("静态代码块");

}

{

age = 0;

System.out.println("非静态代码块");

}

public Person(String name) {

this.name = name;

System.out.println("构造方法");

}

public String toString(){

String str = nationality+", ";

str += name + ", ";

str += age;

returnstr;

}

}

在静态代码块中为静态常量nationality初始化；由于age在创建对象时都是0，因此不需要构造方法为其传递参数，可以在非静态代码块为其初始化；每个对象都有自己的name，因此name必须在构造方法中初始化。

测试类Test类代码如下：

public class Test {

public static void main(String[] args) {

System.out.println("程序开始");

Person p1 = new Person("张三");

Personp2 = new Person("李四");

System.out.println(p1);

System.out.println(p2);

}

}

运行结果如下：

程序开始

静态代码块

非静态代码块

构造方法

非静态代码块

构造方法

中国, 张三, 0

中国, 李四, 0

Test类的主方法执行后，当需要Persen类时，加载类，执行静态代码块，且只执行一次，然后创建对象p1，先执行非静态代码块，再执行构造方法。之后再创建对象p2，又一次执行非静态代码块，和构造方法。

这个例子只是介绍静态代码块、非静态代码块的相关知识，实际上也可以不使用非静态代码块，直接在构造方法中初始化年龄。

## 4.6 包

在开发复杂的系统时，通常会有大量的类，为了更好地组织管理这些类，需要对这些类进行分类分组，将密切相关的类分在一组中。Java 提供了包机制，可将密切相关的类放在一个包中。

### 4.6.1 包的定义

Java使用package关键字来定义包，package关键字后面跟着包名，也就是包的名字（习惯上包名用小写字母）。使用package定义包的这条语句，必须放在Java源程序文件的第一行（当然注释除外）。

例如定义一个名字为p1的包，就要在文件的开始处输入下面一行。

package p1;

定义p1包后，在这个文件中定义的所有类就属于这个包。如果包中的类太多，还可以再细分类，定义子包，如p1.p11、p1.p12等，包名和子包名之间用“.”连接，如定义p1包中的p11子包的代码如下。

package p1.p11;

当然每个文件只能有一个定义包的语句，也就是一个文件中的类肯定是在同一个包中，但在多个文件中可以使用同一个包的定义语句，也就是一个包可以包含多个文件。

例4.11 定义两个包p1和p2，在p1包中定义两个类A和B，在p2包中定义类C。再定义一个子包p1.p11，在该子包中定义类D。

A.java文件中的代码：

package p1;

public class A {

private int a;

int b;

public int c;

}

B.java文件中的代码：

package p1;

public class B {

}

C.java文件中的代码：

package p2;

public class C {

}

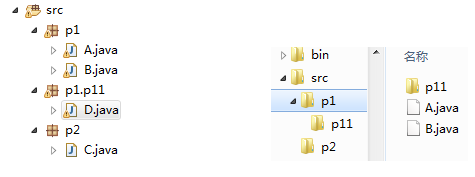
D.java文件中的代码：

package p1.p11;

public class D {

}

对照一下Eclipse中的包和类（图4.4a）与资源管理器中的文件夹和文件（图4.4b），可以发现包与文件夹是对应的，包中的子包对应文件夹中的子文件夹。



a b

图4.4 包与文件夹

本节之前定义的类都没有指定哪个包，他们的类文件实际上是保存在“src”文件夹中，我们称这些没有指定包的类在默认包（或缺省包）中。

有了包的概念，我们就可以在不同的包中定义同名的类，而不会产生混淆。

### 4.6.2 访问其他包中的类

同一个包中的类可以直接使用，而使用其他包中的类则需要告知编译器，使用的是哪个包中的类。可以使用两种方法告知编译器到底使用哪个包中的类，第一种方法是在每次使用时用前缀指定包名，第二种方法是将要使用的类导入到当前文件。

1．使用前缀指定包

使用前缀指定类所在包的格式是“包名.类名”。例如在C类中使用A类可以使用“p1.A”，使用D类可以使用“p1.p11.D”。

例如在C类中增加一个方法fun()，在该方法中创建一个A类对象、一个B类对象和一个D类对象，代码如下。

package p2;

public class C {

public void fun(){

p1.A a = new p1.A();

p1.B b = new p1.B();

p1.p11.D d = new p1.p11.D();

}

}

在每次使用不同包中的类时，都要加上包名作为前缀，如果大量的频繁引用其他包中的类，显然是非常麻烦的，因此我们通常使用导入类的方式使用其他包中的类。

2．使用import导入类

为了避免每次引用其他包中的类都要加包名前缀，可以使用import将类导入到当前文件中。类被导入后，再使用该类时就可以直接引用了。例如可以将上面的C类改成如下的形式。

package p2;

import p1.A;

import p1.B;

import p1.p11.D;

public class C {

public void fun(){

A a = new A();

B b = new B();

D d = new D();

}

}

在文件的开始处用import将类A、类B和类D导入进来，在后面就可以直接使用了。如果一个包中有很多类，也可以用一句import将他们都导入近来，格式是“import 包名.\*”，用星号表示所有类，可以将C类改成下面的形式。

package p2;

import p1.\*;

import p1.p11.D;

public class C {

public void fun(){

A a = new A();

B b = new B();

D d = new D();

}

}

当然对D类的导入不能省略，也就是说“import p1.\*;”只是将p1包中的类全部导入，与他的子类没有关系。如果子类中也有很多类需要引用，也可以使用“import p1.p11.\*;”将该子包中的类全部导入。

3．包访问权限

对于类成员的访问权限，前面已经介绍了public（公有）和private（私有）访问权限，公有成员在任何地方都可以访问，私有成员只有在自己的类中才可以访问。

在上面的A类中，除了公有成员a、私有成员c，还有一个没有指定访问权限的成员b，称b的访问权限为缺省访问权限，Java中的缺省访问权限是在同一个包中可以访问，在不同的包中不可以访问。

例如，我们在与类A同一个包的B类中定义一个方法，在方法中定义一个类A的对象，并访问类A的三个属性，代码如下。

package p1;

public class B {

public void fun(){

A a = new A();

a.a = 10;

a.b = 20;

a.c = 30;

}

}

由于a是类A的私有成员，因此“a.a = 10;”存在语法错误，在类B中不能访问类A的私有成员，而另外两行赋值没有错误。

同样，我们在与类A不在同一个包的C类中定义一个方法，在方法中定义类A的对象，并访问类A的三个属性，代码如下。

package p2;

import p1.\*;

public class C {

public void fun(){

A a = new A();

a.a = 10;

a.b = 20;

a.c = 30;

}

}

由于类A与类C不在同一个包中，前两行赋值语句都有错，只有最后一行赋值语句无错误。

事实上JDK中的类非常多，这些类就是按功能放在不同的包中。如我们常用System类、Math类和String类就在java.long包中。在后面的学习过程中，我们会遇到JDK不同包中的各种类。

## 4.7 对象数组

数组中的元素不仅可以是基本数据类型，也可以是对象，元素是对象的数组也称为对象数组。

下面以学生类为例介绍对象数组的使用方法。

例4.12 设计学生类，然后在测试类中定义学生对象数组，在数组中保存若干学生的信息并输出。

学生类定义如下：

public class Student {

private String no;

private String name;

private int age;

private double score;

public Student(String no, String name, int age, double score) {

super();

this.no = no;

this.name = name;

this.age = age;

this.score = score;

}

public String getNo() {

return no;

}

public void setNo(String no) {

this.no = no;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public double getScore() {

return score;

}

public void setScore(double score) {

this.score = score;

}

public String toString(){

String str = no + ",";

str += name + ",";

str += age + ",";

str += score;

return str;

}

}

测试类Test代码如下：

public class Test {

public static void main(String[] args) {

Student[] s = new Student[5];

s[0] = new Student("20170101", "张三", 16, 620);

s[1] = new Student("20170102", "李四", 17, 680);

s[2] = new Student("20170103", "王五", 20, 650);

s[3] = new Student("20170104", "赵六", 18, 640);

s[4] = new Student("20170105", "冯七", 17, 690);

double score = 0;

for(int i=0; i<s.length; i++){

score += s[i].getScore();

System.out.println(s[i]);

}

System.out.println("平均成绩: " + score/s.length);

}

}

运行结果如下：

20170101, 张三, 16, 620.0

20170102, 李四, 17, 680.0

20170103, 王五, 20, 650.0

20170104, 赵六, 18, 640.0

20170105, 冯七, 17, 690.0

平均成绩: 656.0

首先创建对象数组s，然后再创建每个元素引用的对象，通过循环输出每个对象的信息，并对成绩求和，最终输出平均成绩。

对象数组s在内存中的存储情况如图4.5所示，图中只画出了元素s[0]和s[1]所引用的对象，s[2]、s[3]和s[4]引用的对象未画出。

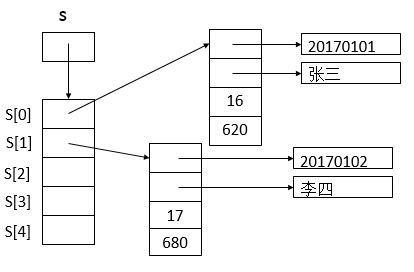


图4.5 对象数组在内存中的存储

## 4.8 作业

### 4.8.1 单项选择题

1. 下列不属于面向对象编程特征的是

A. 封装 B. 指针操作 C. 多态性 D. 继承

2. 下列说法正确的是

A. java中包的主要作用是实现跨平台功能

B. package语句只能放在import语句后面

C. package语句必须放在文件的最前面

D. 可以用#include关键词来标明来自其它包中的类;

3. 关于构造方法，下列说法错误的是

A. 构造方法不可以重载

B. 构造方法用来初始化该类的一个新的对象

C. 构造方法具有和类名相同的名称

D. 构造方法不能指定返回值类型

4. 简单变量和引用变量的初始化有什么区别？

A. 简单变量的初始化需要为它分配一个数值单元，而引用变量的初始化只需要给它赋一个值就可以了

B. 简单变量的初始化只需要给它赋一个值，而引用变量的初始化需要指向一个存在的对象

C. 二者的初始化都需要指向一个存在的单元，但前者需要指向一个数值单元，后者需要指向一个对象

D. 二者都需要初始化，因此它们的初始化也没有区别

5. 关于final变量正确的说法是。

A. 最后一次出现的变量 B. 就是变量的另一种说法

C. 变量在第一次初始化之后值就不能再变 D. 只能在定义的同时进行初始化

6. 下述哪些说法是正确的？

A. 用static关键字声明实例变量 B. static必须与final同时使用

C. 局部变量在方法执行时创建 D. 局部变量在使用之前必须初始化

### 4.8.2 判断对错

1. Java语言中，任何一个类都有构造方法。

2. 使用构造方法只能给实例成员变量赋初值。

3. 类是一种类型，也是对象的模板。

4. 类中说明的方法可以定义在类体外。

5. 在非静态方法中不能引用静态变量。

6. 在静态方法中不能引用非静态变量。

7. 创建对象时系统将调用适当的构造方法给对象初始化。

8. 使用运算符new创建对象时，赋给对象的值实际上是一个引用值。

9. 对象可作方法参数，对象数组不能作方法参数。

10. class是定义类的惟一关键字。

### 4.8.3 编程题

1．编写一个类实现时钟的功能，属性包括时、分、秒，方法有构造方法、一组get和set方法、显示时间的方法，编写Test类对时钟类测试。

2．编写一个类实现复数的运算，要求实现复数加、减、乘等功能，编写Test类对复数类测试。

3．编程创建一个立方体类Box，属性包括立方体的长、宽和高，方法包括构造方法、一组get和set方法、求立方体体积的方法。编写Test类对Box类测试。

4．定义一个学生类（Student），属性包括：学号，班号，姓名，性别，年龄，班级总人数；方法包括：获得学号，获得班号，获得姓名，获得性别，获得年龄，获得班级总人数，修改学号，修改班号，修改姓名，修改性别，修改年龄以及一个toString()方法将Student类中的所有属性组合成一个字符串。定义一个学生数组对象。设计Test类进行测试。

5．定义一个圆类，属性有半径，除了构造方法和set、get方法外，还有一个计算面积和周长的方法。再定义一个圆柱体类，圆柱体类包含一个圆类对象的属性（表示圆柱体底面），还有一个圆柱体高度属性，方法包括计算圆柱体的体积和计算圆柱体的表面积。编写Test类进行测试。